USENIX Association

Summer Conference Proceedings

Atlanta 1986

June 9 - 13, 1986
Atlanta, Georgia USA

For additional copies of these proceedings, write:

USENIX Association

P. O. Box 7

El Cerrito, CA 94530 USA

Price: $25.00 plus $25.00 for overseas mail

ACKNOWLEDGEMENTS

Sponsor:	**The USENIX Association** **P. O. Box 7** **El Cerrito, CA 94350**	
Program Chairperson:	**Mike O'Dell**	
Program Committee:	**John Chambers**	**MCC**
	Mike Hawley	**Lucasfilm, Ltd.**
	Sam Leffler	**Lucasfilm, Ltd.**
	Jim McKie	**Bell Communications Research**
	Dennis Ritchie	**AT&T Bell Laboratories**
	Spencer Thomas	**University of Utah, Computer Science Department**
Tutorial Coordinator:	**Michael Tilson**	**Human Computing Resources Corp.**
USENIX Meeting Planner:	**Judith DesHarnais**	
Vendor Exhibition Manager:	**John Donnelly**	
Conference Hosts:	**Paul Manno**	**Medical Systems Development Corp.**
	Dan Forsyth	

Terry Countryman, Peter Wan, and Lillie Elliot of Medical Systems Development Corporation assisted in the production of these proceedings. Brent Laminack of In Touch Ministries graciously provided assistance in producing the table of contents. Ralph Amiel of Standard Press patiently answered many dumb questions on the production of a volume this size.

To all of the authors: You deserve many thanks and much appreciation. You have made an otherwise impossible task relatively easy by providing the camera-readies in plenty of time to meet the printing deadlines. To those authors who forgot to clean their laser printers before generating their copy: Yes, it does show. What you see is what the camera sees. Please remember next time.

Uwm: A User Interface for X Windows

Michael Gancarz

Ultrix Engineering Group
Digital Equipment Corporation
Merrimack, New Hampshire 03054

ABSTRACT

Uwm is a user-programmable user interface client application for *X* windows. It provides standard functions for moving, resizing, iconifying, raising, and lowering windows, as well as more exotic functions to accommodate specialized window management tasks. These can be bound to mouse buttons and control keys in combinations of your own choosing. *Uwm* also offers pop-up menus of the "slip-off" variety. Menus may contain shell commands, window manager functions, cut buffer strings, and even references to other menus.

Nearly every aspect of *uwm* allows tailoring by the individual user: menu selections and headings, styles of highlighters, font choices, amount of text padding, function contexts, modes of operations, etc., etc. And, if using color hardware, the user has complete control over color choices, a highly subjective area at best. *Uwm* solves the workstation user interface problem in the true UNIX tradition by giving the choices, and ultimately the power, to the user.

Uwm is included in the 4.3 Berkeley distribution along with *X* windows as user-contributed software.

1. Introduction

X is a network transparent windowing system developed at the Massachusetts Institute of Technology that runs under ULTRIX-32 Version 1.2, 4.2BSD and 4.3BSD UNIX. It provides the familiar model of overlapping windows, as well as many services for the graphics applications programmer. *X* display servers run on computers with either color or monochrome bitmap terminals. The server multiplexes user input and output requests to and from various client programs located either on the same machine or elsewhere on the network. While a client normally runs on the same machine as the *X* server it is communicating with, this need not be the case.

A particularly useful feature of *X* is that, unlike most other windowing systems, the user interface is not part of the server but is a separate client program. Hence, you can choose from a variety of user interfaces, and run them singly or in tandem, on a local or remote machine.

Current user interfaces for *X* include *xwm*, *xnwm*, and, of course, *uwm*. All provide provide basic window management functions, such as moving, resizing, stacking and iconifying windows. Each interface is, to a lesser or greater degree, user-

† ULTRIX-32 is a trademark of Digital Equipment Corporation.

‡ UNIX is a trademark of AT&T Bell Laboratories.

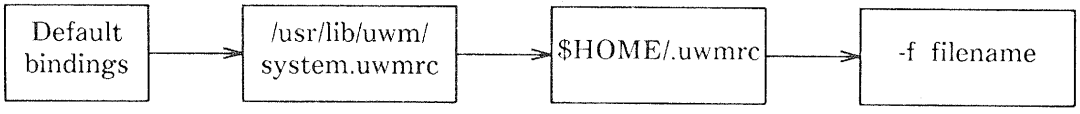
programmable. *Xwm*, the ancestor of both *xnwm* and *uwm*, provides a compact environment with minimum user flexibility and rapid startup. *Xnwm* offers popup menus for window management functions, as well as the ability to tailor mouse button bindings in a dynamic fashion. *Uwm*, too, incorporates popup menus for window management functions, but it carries the popup metaphor one step further by allowing you to specify UNIX commands and "cut buffer" operations in menus. To complement the enhanced menu implementation, *uwm* employs a more sophisticated scheme for specifying mouse button bindings in a variety of contexts.

One design goal of *uwm* was that, for *uwm* to be a truly "user-programmable" interface, it should be capable of emulating nearly every aspect of its *X* predecessors' functional capabilities. In that respect, it succeeded admirably. It is possible for a user to configure *uwm* to perform identically to *xwm*, and you can create popup menus with *uwm* that incorporate the same functions found in *xnwm*'s popups. Differences remain, but they are largely matters of style, not functionality. However, lest you think that we're playing a game of one-upmanship here, remember that you may still run all of these window managers with the others. None of these interfaces is a "loser" and the user wins in every way.

2. What the User Sees

2.1. The Startup File Search Path

You specify mouse button bindings and menu selections for *uwm* with internal defaults and external bindings found in startup files. The defaults and startup files are recognized in the following order:



Uwm writes the internal default bindings out to a temporary file, parses the file, then unlinks the temporary file. While this might seem inefficient, it insures that the internal defaults have been correctly entered by the *uwm* developer by forcing them to pass through the same error-checking mechanisms that the parser uses for all startup files. The temporary file usually gets no farther than the disk buffer cache before the file is unlinked (except on "write-through" file systems).

System administrators can provide for global default window environments with the file */usr/lib/uwm/system.uwmrc*. *$HOME/.uwmrc* contains the bindings and menu specifications for an individual user. Using *-f filename*, you can specify an alternate startup file on the command line for use in, say, a programming or text processing environment. If either of */usr/lib/uwm/system.uwmrc* or *$HOME/.uwmrc* doesn't exist, the program quietly proceeds. A missing *filename* is flagged as an error.

Bindings and menus in the various components of the search path are cumulative. This allows later elements in the chain to take advantage of pre-defined notions of variables, menus and bindings found in earlier files. There are also provisions for overriding anything specified in a previous component.

Uwm uses *yacc(1)* and *lex(1)* to parse the startup files. The parser strips the input of comments, blank lines, and unquoted whitespace, then builds singly-linked lists of menus and mouse button bindings. Illegal mouse binding combinations, references to non-existent menus, and other errors are flagged before the program begins sending commands to the *X* window server. All errors in the startup file specifications are considered fatal. However, to simplify debugging, *uwm* attempts to report all errors it finds before it exits.

2.2. What's in the Startup File

The startup file may contain variables, mouse button/key binding information, and menu specifications. These may be specified in any order. It isn't necessary to declare a menu before referencing it.

2.2.1. Variables

Uwm supports several variable types. These variables generally have a global effect on *uwm* functions. Boolean, numeric and string types are supported. The variables are covered in detail in the *uwm* manual page, so we'll touch on them briefly here.

The boolean variables, if present, declare that functions will generally have certain characteristics. One example is the *freeze* variable. When *freeze* is set, all "rubber-band boxes" used in resizing or moving windows have a solid appearance; otherwise, the boxes flicker during the operation. Most of the boolean variables can be negated by specifying them as "no<variable>". The *grid* variable, when present, causes an expandable grid to used instead of a box when resizing. Thus, *grid* can be overridden by specifying *nogrid*.

Numeric variables are used to fine-tune the positions and movements of screen objects, such as windows and icons. Another use for numeric variables is for denoting the amount of "padding" around text used in icons and menus. *Uwm* lets you tailor the placement of text within popup menus and text-style icons. This becomes especially useful when dealing with the more than several dozen text fonts supplied in the typical *X* implementation.

The most common use of string variables in the startup file is that of specifying fonts. The text fonts used in menus, the text icons supplied by *uwm*, and the font used in a window sizing indicator are all user-programmable. This tends to be a popular feature with users, as font selection is one area where it is impossible to please all the people all the time. *Uwm* responds to this need in a non-dictatorial fashion by saying "Have it your way." For the sake of consistency, however, all menus use the font specified with the *menufont* variable.

2.2.2. Mouse Button/Key Bindings

Mouse button usage under *uwm* is determined by button/key bindings specifications in the startup file search path. It is theoretically possible to tie any built-in window management function to any button combination. *Uwm* owes much of its success (and its criticism) to this feature. You have total control over whether, say, the left mouse button moves windows, the middle button iconifies them, and the right button resizes them. The resultant flexibility, while enormously powerful, has a drawback: The unsuspecting user has no idea what the mouse buttons are used for unless he's using a set of bindings known to him. For this reason, *uwm* is the bane of trade show demonstrators and the joy of experienced users.

The standard format for a mouse/button key binding looks like this:

<function> = [<keyspec>] : [<context>] : <button action> [: <menu>]

The *keyspec*, *context*, and *menu* fields are optional, but you must include the colon separators regardless. Note that the colon before the *menu* field is required only if the *menu* field exists.

Function is any one of the many window management functions provided. There are functions for moving, resizing, iconifying, and stacking windows in a variety of ways. You can move windows in any direction by fixed increments, relative increments, or by manually positioning them. Windows can be resized by "stretching" one or two sides at a time. Icons normally appear in default positions,

but their positions can be changed during "shrink window to icon" operations by using the *f.newiconify* function. An *f.menu* function makes it easy to bind one or more popup menus to a given button combination. All in all, there are more than a dozen different operations you can choose from, and the number keeps growing as people find more creative things to do with windows.

The *keyspec* field denotes which key(s), if any, on the keyboard must be held down during an entire *uwm* operation. Mouse button presses that occur when the mouse cursor is in an application window could possibly go to the client running in the window or to the window manager. *Uwm* sidesteps the potential conflicts by insisting that when one or two user-specified keyboard "control" keys are held down, all button presses are intercepted by the window manager. The keys are chosen from a group of four possibilities, one of which is the LOCK key. This yields nearly a dozen chording opportunities. If the LOCK key is specified, it makes a handy switch for turning the window manager "off" and "on" while working within an application window. But suppose you aren't using any applications that require the use of the mouse buttons? Fine. A null *keyspec* lets you bypass the use of the keyboard keys at the expense of disallowing the use of the mouse buttons within applications windows.

The *context* field pertains to the position of the mouse cursor when a mouse button action has occurred. The window manager assumes that three possible contexts exist: window, icon and the root (background) window. When the cursor is in a normal window--as opposed to an icon window--when a button is pressed or released, then it is considered to be in a window context. Icon and root contexts apply in a like manner. This capability allows you to specify different actions depending on where the mouse cursor is. For example, when the left button is pressed while the cursor is in a window, you may wish to lower the window in the window stack. But if the left button is pressed while the cursor is in the background, then you may want a full-screen refresh to occur instead. A convenient shorthand makes it easier to include multiple contexts in a binding by combining them with the '|' character. Leaving the *context* field blank means that all contexts are valid for a binding. This is often done when you want a popup menu to appear when a button is pressed, regardless of the mouse cursor position.

Button actions are formed by combining a button *(left, middle,* or *right)* with an action of *up, down,* or *delta* type. The *down* and *up* actions are intuitive: They occur when a mouse button is pressed or released, respectively. The *delta* action's purpose isn't apparent at first, but it is useful. If the mouse is moved more than *n* pixels in any direction while the button is held down, then the *delta* action takes place. The most common use of this is in binding two different functions to the same button, one to the *up* action and the other to the *delta* action. For example, if the function bound to the *delta* action is *f.iconify* and the function bound to the *up* action is *f.lower*, then the window will be lowered in the window stack if you simply pushed and released the mouse button without moving the mouse. However, if the mouse is moved *n* pixels before releasing the button, then the window will be iconified. Like nearly everything else in *uwm*, *n* is a user-selectable quantity.

Some subtleties exist regarding the relationship of the *context* field and the *button action* field. When you press a button, the current location of the mouse cursor is read by the window manager to determine the relevant context. Upon releasing the button, the location of the cursor is again read to determine the (possibly changed) context. This is done because the function bound to the button *down* action may have altered the context, i.e., the window or icon may no longer be there after the function has taken effect. A still more useful aspect of this is that the user may move the mouse to a different context for the button *up* action. Consider the following bindings:

```
f.iconify = ctrl : window : left down
f.iconify = ctrl : window : left up
```

The *f.iconify* function "iconifies" a window; i.e., it turns a window into an icon. When the left button is pressed with the cursor within a window, that window is iconified. If the user moves the mouse cursor to a different window and then releases the button, the new window will also be iconified. The result is that you have accomplished two functions with one button click. The visual effect on the screen reminds one of a highly adept sleight-of-hand artist.

The thoughtful reader at this point is probably wondering: What context is used for a *delta* action? For a *delta* action to occur, you recall, the mouse cursor must have been moved *n* pixels. In doing so, the context may have changed. After a fair amount of experimentation (and not a little frustration), it was determined that the proper context for the *delta* action is the context when the button was originally pressed. Early *uwm* users found that the position of the mouse cursor when the *delta* action was actually invoked was too unpredictable to used as the point in time to read the context.

The *menu* field in a mouse button binding is required only when using the *f.menu* function. This function paves the way for user-defined menus bound to favorite binding combinations. When the button action bound to the menu function occurs, a popup menu appears on the screen at the mouse cursor position. Which popup menu appears depends on the name in the *menu* field.

You may specify different mouse button bindings with similar menu names, as well as similar mouse button bindings with different menu names. As an example, consider the following bindings:

```
f.menu=  meta  : root : middle down: "CREATE SMALL WINDOWS"
f.menu=  c|s   : root : right down  : "CREATE SMALL WINDOWS"
f.menu=  c|s   : root : right down  : "CREATE LARGE WINDOWS"
f.menu=  c|s   : root : right down  : "CREATE MANUALLY-SIZED WINDOWS"
```

The first binding shows that pressing the middle mouse button while holding the META key causes the "CREATE SMALL WINDOWS" popup menu to appear. If you select nothing on the menu, the menu disappears and that is that. The last three bindings are an example of *uwm*'s notion of "slip-off" menus. The "CREATE SMALL WINDOWS" popup menu appears when both the CONTROL and SHIFT keys are held down and the right mouse button is pressed. (That may sound confusing, but it's really very simple in practice.) By moving the mouse cursor off the menu without selecting an item, the "CREATE SMALL WINDOWS" menu disappears and the "CREATE LARGE WINDOWS" popup appears in its place. Likewise, the "CREATE MANUALLY-SIZED WINDOWS" popup menu replaces the "CREATE LARGE WINDOWS" menu if no selection is made. The user merely has to brush the mouse to the right to move rapidly from one menu to the next. Hence the name "slip-off" menus. The menus pop into place extremely quickly, even on slow hardware implementations. The reason it works is because, since the user programmed the menus himself, he knows what to expect.

2.2.3. User-programmable Menus

Uwm uses a simple syntax for specifying menus in the startup file. Experience has shown that even unsophisticated UNIX users can create complex menu structures using the commands available. Rather than describe the syntax in detail, let's look at a typical menu:

```
menu "EASY COMMANDS" = {
Move a window:          f.move
Create a window:        !"xterm &"
Edit UWM startup file:  !"xterm =65x80+5+5 -e vi $HOME/.uwmrc &"
Access BIGVAX:          !"xterm =24x80+5+5 -e rlogin bigvax &"
Xterm...:               |"xterm -fn 6x10 -i -fg Blue -bg White "
Clear Screen:           ^clear
More -->:               f.menu: "MORE EASY COMMANDS"
}
```

Phrases appearing before the first colon are what the user sees in the popup menu. The portion after the first colon represents the action to be taken when that menu item is selected. The first item shows how the built-in window manager function, *f.move*, can be invoked via a menu to move a window. In fact, all window manager functions may be placed on menus, if desired. The next three lines show how to create various windows via menu selections: One creates a plain window; another runs the *vi* editor on the window manager startup file (notice the shell variable globbing); the third runs an *rlogin* in a window to another host machine. The "Xterm..." item isn't a UNIX command at all. The '|' symbol causes the text following to be placed in the *X* window server's "cut buffer". The cut buffer's contents can be retrieved at any time and dropped into a terminal emulator window as if the string had been typed on the keyboard. "Clear Screen" is a special instance of a cut buffer operation that appends a newline to the string as it places it in the cut buffer. When the buffer contents are dropped in a window, the UNIX command *clear(1)* is immediately executed. Finally, the "More -->" entry, when selected, erases the current popup menu on the screen and replaces it with yet another menu of commands.

There is tremendous flexibility here. The user can define menus containing items that are most useful for his or her particular operating environment. Window manager functions, UNIX shell commands, text strings, and references to other menus can be freely mixed in any order you wish. Several window managers may be run simultaneously, allowing myriad possibilities--without any knowledge of programming whatsoever.

3. User Interfaces for X-perts

Finding the right interface for every possible user is a formidable task indeed. *X* windows deals with this issue by not dealing with it. It assumes that an applications developer will write his own user interface and run it as a separate client that exists outside the window server itself. However, most users have neither the time nor the desire to create such a client. Those who have a strict user interface definition will undoubtedly take the time to do so. The rest will probably use *uwm* until something better comes along.

Uwm embodies the same user interface philosophy that *X* windows does, only at a higher level, i.e., you can't please all the people all the time. But if you let all the people at least have a say in what pleases them, you're well on your way to providing the ideal. *Uwm* gives the user plenty of opportunities for choices. This makes it easy to find an interface that works for him. It may not work well for the next guy, but he'll be quite content with his own environment *because he created it.*

Most of the complaints we've received about *uwm* to date are of the "Why can't I do this or that?" variety. Usually, you can do this or that and much more. But it takes a certain amount of experience with the program to realize its potential. In the interest of speeding up that realization, Appendix A is a copy of the

author's own startup file, edited for brevity. It contains numerous examples of mouse button/key bindings and menus used daily in a software engineering environment. Some of the menus in the example also use color specifications for individual menu items. See the *X* documentation for details on using color menus in *uwm*, as adequate coverage would require more space than we have here.

The button/key bindings are largely self-explanatory, given a little study. You can guess at what the functions do, as their actions are not so important as the way they are bound. Again, notice that binding different functions to button down and button up lets you combine multiple capabilities on a single button. By pressing a second button before releasing the first, you can abort the button up operation. Also, the ordering of the *f.menu* bindings is significant. That is the order in which they will appear in the "slip-off" sequence.

Smart folks bind the most often used window management functions to button/key combinations and relegate the lesser-used functions to menu items. That way you can invoke them quickly. *Uwm* is perhaps the fastest window manager on today's workstations because of that simple capability.

Most of the menu specifications are straightforward, although some menu selections tend to be a bit creative. One instance is that of the "STDFILE" notion. The STDFILE menu contains an entry that stashes a text string, usually a filename, in a file called *STDFILE*. Other entries in the STDFILE menu can be used to operate on that text string. The possibilities for creating a dynamic command invocation mechanism here are obvious.

The "HANDY COMMANDS" menu contains a potpourri of commands that are usually too painful to type manually. It contains a call to a "SHUTDOWN: Are you sure?" menu which queries the user before running shutdown to prevent accidents. A calendar window is available by running a "browser" program on *cal(1)*. The browser program is really a shell script that *exec's* its arguments, then waits for a RETURN to be typed on the keyboard. The "=67x38+2+2" modifier places the calendar at the upper left hand corner of the screen.

Xset(1) is a program that changes various characteristics of the screen interface, such as the background color, keyboard click volume, mouse acceleration, and so on. The numerous references to *xset* show how easy it is to change those characteristics via menu items.

There is much more that can be done in startup files that hasn't even been explored yet. For example, it should be possible to create menu entries that would replace the current startup file with a different one, then restart *uwm* with the new set of bindings. The ability to do shell "globbing" with a menu command bears some experimentation, as do self-modifying startup files. When you combine normal UNIX text processing tools with startup files, the potential increases by an order of magnitude, to say the least. Appendix B contains a shell script that uses *awk(1)* and *ctags(1)* to generate a startup file containing menus which contain names of all functions in a directory full of C source files. Selecting a menu item causes an editor window to be opened with the cursor sitting on the C function in the correct file.

4. Things We'd Like to See Dept.

While *uwm* does a fair amount of work as distributed, there are some features that time did not allow us to put in. Those with source code may feel inclined to make changes. *Uwm* was designed with that goal in mind. Following are some suggestions for those with the urge to hack.

Double Click Action

The mouse button *up*, *down*, and *delta* actions supported suffice for most cases. But just as programmers benefit from more RAM, *uwm* users would like yet another button action. *Xlib(3)* doesn't currently support the kind of lookahead mechanisms needed to do this well. The window manager would have to maintain its own event queue.

Bitmap Menu Items

Text menu items work adequately in most environments. But once in a while it is desirable to add a graphic or two to a menu item for effect.

Dynamic Button/Key Bindings

It would be useful to be able to modify the bindings on the fly and change the run-time environment dynamically. The possibilities for confusing the user would be limitless. Some sort of visual feedback would help.

Alternate Menu Styles

While the built-in "slip-off" menus have proven to be very efficient, why not add a modifier to the menus which would denote an alternate style of display? Pull-downs, dropdowns, and horizontal popups should be fairly easy to implement. Some people have also asked for the slip-off menus to be "reversible". Currently, if you accidentally fall off a menu, you must restart the sequence to return to the original menu. A minor annoyance, but a fix is justifiable.

5. Acknowledgements

The author is deeply grateful to the following individuals and organizations: Paul Asente, DECWRL & Stanford University, for ideas contributed indirectly through *xnwm(1)*; Tony Della Fera, MIT Project Athena & DEC, for ideas borrowed from *xwm(1)*; Jim Gettys, MIT Project Athena & DEC, for general suggestions and testing; Rich Hyde, DEC Ultrix Engineering Group, for parser contributions; Bob Scheifler, MIT Laboratory for Computer Science, for numerous bug reports; members of the Ultrix Base Workstations Group at DEC Merrimack for their day-to-day testing and support; and Jesus Christ, Heaven, for answering my prayers on how to pull it all together.

6. References

James D. Foley, Andries Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company, Reading, Mass., 1982

Michael J. Hawley, Samuel J. Leffler, *Windows for UNIX at Lucasfilm*, Usenix Proceedings, Summer, 1985

Jim Gettys, Ron Newman, Tony Della Fera, *Xlib - C Language X Interface*, MIT Project Athena, 1986

Bob Scheifler, *X* manual page, MIT-LCS, 1985

APPENDIX A - Sample Startup File

```
#
# GLOBAL VARIABLES
#
resetvariables;resetbindings;resetmenus
#reverse
autoselect
delta=25
freeze
grid
normali
nonormalw
hiconpad=5
hmenupad=6
iconfont=oldengssx
menufont=timrom12b
push=1
pushabsolute
resizefont=helv12b
viconpad=5
vmenupad=3
volume=4
#zap
maxcolors=0

#
# BUTTON-KEY BINDINGS
#
# FUNCTION    KEYS CONTEXT    MOUSE BUTTON ACTIONS
f.newiconify=     meta  : window|icon : left delta
f.lower=          meta  : window|icon : left up
f.iconify=        meta  : window      : middle delta
f.iconify=        meta  : icon        : middle down
f.iconify=        meta  : window|icon : middle up
f.moveopaque=     meta  : window|icon : right down
f.raise=          meta  : window|icon : right up
f.circledown=     meta  : root        : left down
f.circleup=       meta  : root        : right down
f.newiconify=     lock  : window|icon : left delta
f.lower=          lock  : window|icon : left up
f.iconify=        lock  : window|icon : middle down
f.move=           lock  : window|icon : right down
f.circleup=       lock  : root        : left down
f.beep =          lock  : root        : middle down
f.circledown=     lock  : root        : right down
f.menu=           m|s   :             : left down   :"WINDOW OPS"
f.menu=           m|s   :             : left down   :" MISCELLANEOUS "
f.menu=           m|s   :             : left down   :" XTERMS"
f.pushleft=       m|s   : window|icon : middle down
f.pushright=      m|s   : window|icon : right down
f.pushup=         m|c   : window|icon : middle down
f.pushdown=       m|c   : window|icon : right down
```

```
#
# MENU SPECIFICATIONS
#
menu = "WINDOW OPS" (White:Black:Black:White) {
"(De)Iconify":(White:"#2f0f2f"):        f.iconify
Move:(White:"#4f0f4f"):                 f.move
Resize:(White:"#6f0f6f"):               f.resize
Lower:(White:"#8f0f8f"):                f.lower
Raise:(White:"#af0faf"):                f.raise
Solid Move:(White:"#cf0fcf"):           f.moveopaque
"Others        -->":(White:"#cf0fcf"): f.menu:"EXTENDED WINDOW OPS"
}

menu = "EXTENDED WINDOW OPS" (White:Black:Black:Turquoise) {
Iconify at New Position:(White:"#ff0000"):          f.newiconify
Focus Keyboard on Window:(White:"#d00000"):         f.focus
Freeze Server:(White:"#b00000"):                    f.pause
UnFreeze Server:(White:"#900000"):                  f.continue
Circulate Windows Up:(White:"#700000"):             f.circleup
Circulate Windows Down:(White:"#500000"):           f.circledown
Refresh Entire Screen:(White:"#300000"):            f.refresh
}

menu = " XTERMS" (White:Black:Red:White) {
*Local*:(White:Black): !"xterm =80x24+0+0 -n local&"
Antic:(White:Blue): !"xterm =80x24+0+0 -s -e antic&"
Decvax:(White:Black): !"xterm =80x24+0+0 -s -e decvax&"
"Unsized          -->":(White:"#222222"): f.menu:"UNSIZED XTERMS"
"Full-sized       -->":(White:"#666666"): f.menu:"LONG XTERMS"
}

menu = "LONG XTERMS" {
*Local*: !"xterm =80x65+0+0 -n Local -bw 5 -bd Black -cr Black&"
Antic: !"xterm =80x65+0+0 -s -bw 5 -bd Blue -cr Blue -e antic&"
Decvax: !"xterm =80x65+0+0 -n Decvax -s -e decvax&"
}

menu = "UNSIZED XTERMS" {
*Local*: !"xterm -n local -bw 5 -bd Black -cr Black&"
Antic: !"xterm -s -bw 5 -bd Blue -cr Blue -e antic&"
Decvax: !"xterm -s -bw 5 -bd Black -cr Black -e decvax&"
}

menu = " MISCELLANEOUS " (White:Black:Black:White){
"Cut Buffer Strings -->":(White:SkyBlue): f.menu:"CUT BUFFER STRINGS"
"Handy Commands    -->":(White:CornflowerBlue): f.menu:"HANDY COMMANDS
"STDFILE Menu      -->":(White:SkyBlue): f.menu:"'STDFILE' MENU"
"User Preferences -->":(White:CornflowerBlue): f.menu:"PREFERENCES"
}

menu = "'STDFILE' MENU" (White:Black:Yellow:Black) {
Edit STDFILE: !"xterm -i =80x65+5+5 -e vi `cat $HOME/STDFILE`&"
Set & Edit STDFILE: !"xterm -i =80x65+5+5 -e esetstdfile&"
Set STDFILE: !"xterm -i =40x3+100+100 -e setstdfile&"
```

```
}

menu = "CUT BUFFER STRINGS" (White:Black:White:Red) {
"Make > tmp &":(Black:White):      ^"rm -f tmp;make >tmp&"
"tail -f tmp":(Black:White):       ^"clear;tail -f tmp"
"Xted":(Black:White):              ^"xted -i &"
"biff n":(Black:White):            ^"biff n"
"xterm -fn 9x15 ...":(Black:White): |"xterm -fn 9x15 "
"Clear Screen":(Black:White):      ^clear
}

menu = "HANDY COMMANDS" (White:Black:Black:White) {
Edit .uwmrc: !"xterm -i =80x65+5+5 -e vi $HOME/.uwmrc&"
Restart Window Manager: f.restart
Exit Window Manager: f.exit
Vi Editor: !"xterm -i =80x65+5+5 -e vi&"
Mail Window: !"xterm =+0+0 -e rsh antic mail&"
Restart X Server: f.menu:"RESTART SERVER: Are you sure?"
Shutdown:f.menu: "SHUTDOWN: Are you sure?"
Xted Editor: !"xted -i &"
1986 Calendar: !"xterm =67x38+2+2 -n 1986 -e browse cal 1986&"
}

menu = "SHUTDOWN: Are you sure?" (White:Black:White:Red) {
No:(Black:White):     !""
Yes:(Black:White):    !"sudo /etc/shutdown -h now"
}

menu = "RESTART SERVER: Are you sure?" (White:Black:White:Red) {
No:(Black:White):     !""
Yes:(Black:White):    !"$HOME/bin/restartX&"
}

menu = "PREFERENCES" (White:Black:White:Blue) {
Bell Loud:            !"xset b 7&"
Bell Normal:          !"xset b 3&"
Bell Off:             !"xset b off&"
Mouse Fast:           !"xset m 4 2&"
Mouse Normal:         !"xset m 2 5&"
Mouse Slow:           !"xset m 1 1&"
}
```

APPENDIX B

```
#
# func.mm -- "menu maker" script for 'uwm' window manager
#            Uses ctags to create a series of menus for a uwm startup
#            file.  Selecting an item off the menu invokes 'vi -ta'
#            on the appropriate function.  The slip-off menus are bound
#            to 'ctrl|shift' on the left button in any context.
#
#            To change the bindings, alter the print statements in the
#            awk portion.
#

FILE=functions.uwmrc
MENUSIZE=10
if test -s $FILE ; then uwm -f $FILE
else
ctags *.c
cat >awktmp <<!
BEGIN {
print "resetvariables;resetbindings;resetmenus;autoselect"
print "delta=25;freeze;grid;hiconpad=5;hmenupad=6"
print "iconfont=oldeng;menufont=timrom12b;resizefont=helv12b"
print "viconpad=5;vmenupad=3;volume=7;zap"
print "f.menu=c|s::left down:\"EDIT FUNCTIONS\""
print "menu=\"EDIT FUNCTIONS\"(White:Black:White:Red){"
i = 2
}
{ if (NR > $MENUSIZE) {
        print "}"
        printf "f.menu=c|s::left down:\"EDIT FUNCTIONS #%d\"\n", i
        printf "menu=\"EDIT FUNCTIONS #%d\"(White:Black:White:Red){\n", i
        NR = 1
        ++i
   }
}
{printf "%s:(Black:White):!\"xterm =80x65+0+0 -e vi -ta %s&\"\n",\$1,\$1}
END {print "}"}
!
awk -f awktmp tags >>functions.uwmrc
rm -f awktmp &
uwm -f functions.uwmrc
fi
```